6. □□□□
```python
def closing_ceremony():
print("□□□□□□□□□□□□□□□□□□□")
time.sleep(2)

# □□□□□
music_player.play("closing_song.mp3")

# □□□□□□□□□□
robot_controller.perform("finale")

# □□□□
audience_interaction.cheer()

time.sleep(5) # □□□□□ 5 □
print("□□□□2025 □□□□□□□□□□□□□□□□□□□□")
```

7. □□□
```python
def main():
# □□
opening_ceremony()

# □□
intermission()

# □□
dance_session()

# □□
closing_ceremony()

if __name__ == "__main__":
main()
```

8. □□□□
```python
# □□□□□
main()
```

9. □□□□
- **RobotController**: □□□□□□□□□□□□□□

```
# □□□□□□□
with multiprocessing.Pool(4) as pool:
pool.apply_async(self.orchestra.play, (opener,))
pool.apply_async(self.light_system.start_sequence)
pool.apply_async(self.dance_robots.initialize_pose)

# □□□□□□□□
def dance_session(self, music_list):
conductor = ConductorAI(tempo_detect_mode="real-time")
for piece in music_list:
# □□□□□□□□
score = MelodyComposer().arrange(
piece,
orchestra_size=32,
complexity=0.8
)

# □□□□□□□□□
conductor.load_score(score)
beat_pattern = conductor.analyze_beat()

# □□□□□□
processes = [
multiprocessing.Process(target=self.orchestra.play, args=(score,)),
multiprocessing.Process(target=self.dance_robots.perform,
args=(beat_pattern, "waltz")),
multiprocessing.Process(target=self.light_sync,
args=(score.tempo,))
]

# □□□□□□□
if self.audience_sensor.excitement_level > 0.7:
processes.append(multiprocessing.Process(
target=self.audience_interaction_mode))

[p.start() for p in processes]
[p.join() for p in processes]

# □□□□□□
def light_sync(self, bpm):
color_map = {
"waltz": ["#FF69B4", "#4B0082"],
"tango": ["#8B0000", "#000000"],
"quickstep": ["#00FF7F", "#32CD32"]
}
self.light_system.set_pattern(
bpm=bpm,
```

```
color_theme=color_map.get(self.current_dance_style),
strobe_intensity=self.audience_sensor.get_real_time_data('clapping')
)

# □□□□
def finale(self):
# □□□□□□□□
finale_music = MelodyComposer().combine(
motifs=["ode_to_joy", "blue_danube"],
transition="crossfade"
)

# □□□□□□□
self.dance_robots.finale_choreography(
formation="spiral",
speed=0.8,
led_effect="golden_sparkle"
)

# □□□□□□
self.orchestra.fade_out(duration=15)
self.light_system.sunset_effect(duration=20)

# □□□□□□□□□□□
class DanceSwarm:
def __init__(self, num_robots):
self.robots = [DanceAI(model="H1_v2") for _ in range(num_robots)]
self.sync_controller = MotionSyncMaster()

def perform(self, beat_pattern, dance_type):
# □□□□□□
choreo = DanceAI.generate_choreography(
dance_type,
complexity=3,
spatial_constraints="stage_12m"
)

# □□□□□□
trajectories = self.sync_controller.calculate_formation(
formation_type="dynamic_swarm",
collision_buffer=0.5
)

# □□□□□□
for robot, path in zip(self.robots, trajectories):
robot.execute_movement(
path,
```

```
force_feedback=True,
music_sync=beat_pattern
)
robot.express_emotion(
intensity=self.audience_sensor.current_excitement
)

# □□□□□□□□□□□
class MelodyComposer:
def generate_opening(self, duration, instruments):
# □□ LSTM □□□□□□□
return {
"tempo": 108,
"key": "C_major",
"structure": [
{"measure":1-8, "instrument":"strings", "dynamics":"pp"},
{"measure":9-16, "instrument":"brass", "dynamics":"mf"},
{"measure":17-24, "instrument":"full_orchestra", "dynamics":"ff"}
]
}

# □□□□
if __name__ == "__main__":
concert = ConcertSystem()
concert.opening_ceremony()

# □□□□□□□□□□□
program = [
"Blue Danube Waltz",
"Carmen Suite",
"Swan Lake Suite"
]

concert.dance_session(program)
concert.finale()
```

□□□□□□□□□[7](https://sghexport.shobserver.com/html/baijiahao/2023/07/07/1069307.html)[9](https://www.zghy.org.cn/item/682320743792300032)[14](https://3g.china.com/act/news/10000169/20250129/47918666.html)□□

1. **□□□□□□□**
□□□□□ NTP-PTS □□□□□□□□□□□□
- □□□□□□□□□□<3ms
- □□□□□□□□□<50ms
- □□□□□□□□<200ms

2. **□□□□□□□**
□□ OpenPose □□□□□□□□□□□
```python
DanceAI.move_selection_algorithm(
current_pose,
target_position,
constraints={
'torque_limits': [360, 180, 90], # □□□□□□□
'energy_efficiency': 0.85,
'aesthetic_score': 0.92
}
)
```

3. **□□□□□□**
□□ Transformer-XL □□□□□□□□□
```python
music_generator.predict_next_note(
previous_notes=128,
style_embedding=[0.7, 0.2, 0.5], # □□/□□/□□□□□□
emotion_vector=audience_sensor.emotion_output
)
```

4. **□□□□□□□□□**
□□□□□□□□□□□□□□□
```python
class AudienceFeedback:
def __init__(self):
self.audio_analyzer = ClapDetector(sensitivity=0.7)
self.visual_analyzer = OpenCVEmotionRecognition()
self.thermal_sensor = InfraredHeatmap()

def get_excitement_level(self):
return 0.3*self.audio_intensity +
0.5*self.visual_engagement +
0.2*self.thermal_density
```

□□□□□□□□□□□□
- □□`program.append()`□□□□□□□
- □□`color_map`□□□□□□□□□
- □□□□□□ AI □□□□□□□□

□□□□□□□□□□□□□□□□ SDK□DMX □□□□□□□□□□□□□□□□□ ROS2 □□□□□□□□□□□□□□

```
self.is_dancing = False
print(f"{self.name} □□□□")

def start_singing(self, song):
self.is_singing = True
print(f"{self.name} □□□□ {song}")

def stop_singing(self):
self.is_singing = False
print(f"{self.name} □□□□")

# □□□□□□□
class BandMember(Robot):
def __init__(self, name, instrument):
super().__init__(name, RobotType.BAND_MEMBER)
self.instrument = instrument

def play_instrument(self, song):
print(f"{self.name} □□□ {self.instrument} □□ {song}")

def stop_playing(self):
print(f"{self.name} □□□□")

# □□□□□□□
class MusicDanceParty:
def __init__(self):
self.robots = []
self.band_members = []
self.current_phase = DancePartyPhase.OPENING
self.audience_interaction = False

def add_robot(self, robot):
self.robots.append(robot)

def add_band_member(self, band_member):
self.band_members.append(band_member)

def start_opening(self):
print("=== □□□□□□ ===")
# □□□□
print("□□□□□□□□")
# □□□□□□
print("□□□□□□□")
for member in self.band_members:
member.play_instrument("□□")
# □□□□□□□□□
for robot in self.robots:
```

```
if robot.robot_type == RobotType.DANCER:
robot.start_dancing()
# □□□□□□□□□
print("□□□□□□□□□□□□□□□")
# □□□□□□□
print("□□□□□□□□□□□")
self.current_phase = DancePartyPhase.OPENING

def start_midfield(self):
print("=== □□□□□□ ===")
# □□□□□□□□□□
print("□□□□□□□□□□□□□□□□□□□□")
# □□□□□□□□
world_famous_songs = ["□□□□□", "□□□□", "□□□□□"]
song = random.choice(world_famous_songs)
print(f"□□□□□□ {song}")
for member in self.band_members:
member.play_instrument(song)
# □□□□□□□□
choir_robots = [r for r in self.robots if r.robot_type == RobotType CHOIR]
solo_robot = random.choice([r for r in self.robots if r.robot_type ==
RobotType.SOLOIST])
for robot in choir_robots:
robot.start_singing(song)
solo_robot.start_singing(song)
# □□□□
self.audience_interaction = True
print("□□□□□□□□□□□□□□□")
self.current_phase = DancePartyPhase.MIDFIELD

def start_closing(self):
print("=== □□□□□□ ===")
# □□□□□□□□□
print("□□□□□□□□□□□□□□□□□□□□□□□")
# □□□□□□□
print("□□□□□□□□□")
for member in self.band_members:
member.play_instrument("□□□")
# □□□□□□□□□□
for robot in self.robots:
if robot.is_dancing:
robot.stop_dancing()
if robot.is_singing:
robot.stop_singing()
# □□□□
print("□□□□□□□□□□□□□")
self.current_phase = DancePartyPhase.CLOSING
```

```
print("□□□□□□□□□□□□")
music_player.play("opening_overture.mp3")
robot_controller.start_dance("opening_dance_routine")
audience_interaction.cheer()
time.sleep(5)
print("□□□□□□□□□□□□□□□□□□")
```

4. □□□□
```python
def intermission():
print("□□□□□□□□□□□□□□□□□□□□□□□□")
music_player.play_random(band)
robot_controller.start_dance("intermission_dance_routine")
audience_interaction.clap()
time.sleep(5)
print("□□□□□□□□□□□□□□□□□□□□□□□□")
```

5. □□□□□□
```python
def dance_session():
print("□□□□□□□□□□□□□□□□")
music_player.play("world_famous_dance.mp3")
robot_controller.start_dance("world_famous_dance_routine")
audience_interaction.cheer()
time.sleep(10)
print("□□□□□□□□□□□□□□□□□□□□")
```

6. □□□□□□
```python
def band_solo():
print("□□□□□□□□□□□□")
solo_instrument = random.choice(band)
print(f"□□□□{solo_instrument}□□□□□")
music_player.play_solo(solo_instrument)
robot_controller.start_dance("solo_dance_routine")
audience_interaction.clap()
time.sleep(5)
print("□□□□□□□□□□□□□□□□□□□")
```

7. □□□□
```python
def closing_ceremony():
print("□□□□□□□□□□□□□□□□□□□")
```

```
self.timeline = {
"prelude": {"start": 0, "duration": 300,
"modules": ["lighting", "orchestra", "robot_dance"]},
"dance_session": {"start": 300, "duration": 1800,
"modules": ["ai_music", "crowd_interaction"]}
}

def execute_event(self, event_name):
event = self.timeline[event_name]
# □□□□□□□□□□□□ [8](https://www.zghy.org.cn/item/682320743792300032)
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
LightingSystem.set_pattern(event["light_pattern"])
Orchestra.play(event["score_path"])
RobotGroup.execute_choreography(event["dance_sequence"])

def emergency_stop(self):
# □□□□□
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
RobotGroup.emergency_stop()
Orchestra.fade_out(2.0)
LightingSystem.strobe_alert()
```

□□AI □□□□□□TensorFlow □□□

```python
# □□ Transformer □□□□□□ [3](https://www.sohu.com/a/840700213_161623?scm=10001.325_13-109000.0.0.5_32)[6](https://sghexport.shobserver.com/html/baijiahao/2023/07/07/1069307.html)
class MusicGenerator(tf.keras.Model):
def __init__(self, vocab_size=128):
super().__init__()
self.embedding = layers.Embedding(vocab_size, 512)
self.transformer = Transformer(
num_layers=6, d_model=512, num_heads=8, dff=2048)
self.output_layer = layers.Dense(vocab_size)

def generate(self, prompt, length=512):
# □□□□□□□□□□□ MIDI □□
[12](https://blog.csdn.net/yong7464553/article/details/6077948)
generated = prompt
for _ in range(length):
predictions = self(generated[:, -512:])
next_note = tf.random.categorical(predictions[:,-1,:], 1)
generated = tf.concat([generated, next_note], axis=-1)
return midi_encoder.decode(generated[0]().numpy())
```

□□□□□□□□□□□□ROS□□□
```cpp
// □□ROS□□□□□□□ [8](https://www.zghy.org.cn/item/682320743792300032)[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
void executeDance(const std::string& music_bpm){
// □□□□□□□□□□
DanceMoveLibrary moves = downloadMoves(music_bpm);

// □□□□□□ [11](https://www.bilibili.com/video/av970410235/)
TrajectoryGenerator generator;
auto trajectory = generator.generate(
moves["waltz"],
BPMCalculator.get(music_analysis)
);

// □□□□□□□□
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
SwarmController::syncRobots(
robot_ids,
trajectory,
sync_tolerance=0.05s
);
}
```

□□□□□□□□□□□MIDI□□□□□
```python
# □□□□□□□□□ [12](https://blog.csdn.net/yong7464553/article/details/6077948)
def orchestrate(solo_instrument, style="classical"):
# □□□□□□□□
[6](https://sghexport.shobserver.com/html/baijiahao/2023/07/07/1069307.html)
harmony = AIHarmonizer.generate(
solo_melody,
style=style,
ensemble_size=30
)

# □□□□□□
[1](https://max.book118.com/html/2025/0209/8125137110007030.shtm)
for part in harmony.parts:
midi_out.send(
channel=part.instrument.midi_channel,
note=part.note,
velocity=dynamic_map[part.dynamic]
)
```

```
# □□□□□ (https://www.sohu.com/a/840700213_161623?scm=10001.325_13-109000.0.0.5_32)
if style == "jazz":
add_improvisation(solo_instrument)
```

□□□□□□□□□□□□□+□□□□

```python
# □□□□□□□□ (https://www.sohu.com/a/840700213_161623?scm=10001.325_13-109000.0.0.5_32)[9](https://m.sohu.com/a/695871417_391452/)
class AudienceInteraction:
def __init__(self):
self.emotion_model = load_emotion_detector()
self.audio_analyzer = RealtimeAudioAnalysis()

def adjust_performance(self):
# □□□□□□□□□□□□
[6](https://sghexport.shobserver.com/html/baijiahao/2023/07/07/1069307.html)
emotion_score = self.emotion_model.calculate(camera_feed)

# □□□□□□□□□□□□□
(https://www.diyifanwen.com/jiaoan/youeryuandabanyishujiaoan/07102616403816148301.htm)
applause_level = self.audio_analyzer.get_applause()

# □□□□□□
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
Orchestra.set_tempo(
base_tempo * (1 + applause_level*0.1)
)
LightingSystem.set_brightness(
emotion_score["excitement"] * 100
)
```

□□□□□□□□TTS+NLG□

```python
# □□□□□□ [4](https://news.qq.com/rain/a/20250101A06GBS00?media_id&suid)[9](https://m.sohu.com/a/695871417_391452/)
class Announcer:
def __init__(self):
self.tts = pyttsx3.init()
self.nlg = GPT3_Interface()

def announce(self, program_info):
```

```
# □□□□□□□□□ [4](https://news.qq.com/rain/a/20250101A06GBS00?
media_id&suid)
script = self.nlg.generate(f"""
□□□□□□□□□□□□
□□□{program_info['title']}
□□□□{program_info['composer']}
□□□{program_info['features']}
""")

# □□□□□□□□□□ [9](https://m.sohu.com/a/695871417_391452/)
self.tts.setProperty('rate', 150)
self.tts.say(script)
self.tts.runAndWait()
```

□□□□□□□□

```python
if __name__ == "__main__":
# □□□□□□□
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)
controller = ConcertController()
announcer = Announcer()
audience_system = AudienceInteraction()

# □□□□□□
[1](https://max.book118.com/html/2025/0209/8125137110007030.shtm)
try:
announcer.announce_opening() # □□□□□
[4](https://news.qq.com/rain/a/20250101A06GBS00?media_id&suid)
controller.execute_event("prelude") # □□□□

while not performance_ended:
audience_system.adjust_performance() # □□□□□□
[5](https://www.diyifanwen.com/jiaoan/youeryuandabanyishujiaoan/0710261640
3816148301.htm)

if should_transition():
controller.transition_next() # □□□□
[8](https://www.zghy.org.cn/item/682320743792300032)

except EmergencyException:
controller.emergency_stop() # □□□□
[13](https://3g.china.com/act/news/10000169/20250129/47918666.html)

finally:
controller.execute_ending() # □□□
[1](https://max.book118.com/html/2025/0209/8125137110007030.shtm)
```

```python
# Initialize the robot controller
robot_controller = RobotController()

# Initialize the music player
music_player = MusicPlayer()

# Initialize the audience interaction system
audience_interaction = AudienceInteraction()

# Initialize the band
band_size = 30
band = [f"Musician {i+1}" for i in range(band_size)]
```

3. Opening procedures
```python
def opening_ceremony():
Print ("Announcer: Welcome to the 2025 Intelligent Robot Music Ball!" )
time.sleep(2)
Print ("Announcer: Please enjoy the opening overture!" )
time.sleep(1)

# Play the opening overture
music_player.play("opening_overture.mp3")

# Robots start dancing
robot_controller.start_dance("waltz")

# Audience interaction
audience_interaction.cheer()

Time.sleep(5) # Overture lasts 5 seconds.
Print ("Announcer: The opening overture is over and the dance officially begins!"
" )
```

4. Midfield program
```python
def intermission():
Print ("Announcer: It's halftime, please relax!" )
time.sleep(2)

# Play relaxing background music
music_player.play("intermission_music.mp3")

# Robots perform simple interactive performances.
```

robot_controller.perform("interaction")

# Audience interaction
audience_interaction.clap()

Time.sleep(10) # The intermission lasts for 10 seconds.
Print ("Announcer: The intermission is over, and then the climax of the dance!" ")
```

### 5. Dance program
```python
def dance_session():
Print ("Announcer: Here is the dance music, a world famous song!" )
time.sleep(2)

# Conductor starts shooting.
Print ("Command: 1, 2, 3, go!" )
time.sleep(1)

# Play dance music
music_player.play("world_famous_dance.mp3")

# Robots start dancing
robot_controller.start_dance("tango")

# Audience interaction
audience_interaction.cheer()

Time.sleep(10) # Dance music lasts 10 seconds.
Print ("Announcer: Dance music is over, thank you for your enthusiastic participation!" )
```

6. Closing procedure
```python
def closing_ceremony():
Print ("Announcer: Thank you for your participation. It's the closing ceremony!" )
time.sleep(2)

# Play the closing song
music_player.play("closing_song.mp3")

# The robot performs the final performance
robot_controller.perform("finale")

# Audience interaction

```
audience_interaction.cheer()

Time.sleep(5) # The closing song lasts 5 seconds.
Print ("Announcer: The 2025 Intelligent Robot Music Ball has come to a successful conclusion, thank you!" )
```

7. Main program
```python
def main():
# Opening
opening_ceremony()

# Midfield
intermission()

# Dance music
dance_session()

# Closing
closing_ceremony()

if __name__ == "__main__":
main()
```

8. Run the program
```python
# Run the main program
main()
```

9. Code Description
-**RobotController**: controls the dance and performance of intelligent robots.
-**MusicPlayer**: controls the playback of music.
-**AudienceInteraction**: Handle audience interaction, such as applause and cheers.
-**band**: A member of an analog band.

10. Extended functions
-* * More dance music * *: The function dance_session can be extended to support more dance music types.
-* * More robot dances * *: ` robot_controller' can be extended to support more dance types.
-* * Audience interaction * *: audience_interaction can be extended to support more interactive ways, such as voting and grading.

This code example is a simplified version, which may need to be adjusted and expanded according to specific needs in practical application. ●
According to the process requirements of music dance, a set of modular program code framework is designed by combining artificial intelligence and automatic control technology. The following system includes three core modules: robot dance arrangement, intelligent music generation and multi-device collaborative control. Citation [3] (https://blog.csdn.net/weixin _ 42511080/article/details/115808426) [7] (https://sghexport.shaobserver.com/html/baijiaohao/2023/07/00 (https://www.zghy.org.cn/item/682320743792300032) [12] (https://www.bilibilibili.com/video/av970410235/) [14] (https://3g.china

```python
# -*- coding: utf-8 -*-
# Music Ball Master Control System v3.0
import time
import multiprocessing
from pygame import mixer
from robot_dance_ai import DanceAI
from music_generator import MelodyComposer
from stage_control import LightSystem, ConductorAI

class ConcertSystem:
def __init__(self):
Self. orchestra = orchestra controller () # 30-person band control
Self. dance _ robots = dance swarm (16) # Dance robot cluster
Self.light _ system = lightsystem () # Intelligent lighting system
Self. audience _ sensor = audience feedback () # audience emotion perception

# Opening procedure
def opening_ceremony(self):
# Speech synthesis announcement
Self.tts_announce ("Dear guests, the artificial intelligence symphony dance begins now!" )

# Symphony prelude generation
opener = MelodyComposer(style="symphonic").generate_opening(
duration=180,
instruments=["violin","piano","trumpet"]
)

# Multi-device collaborative startup
with multiprocessing.Pool(4) as pool:
pool.apply_async(self.orchestra.play, (opener,))
pool.apply_async(self.light_system.start_sequence)
pool.apply_async(self.dance_robots.initialize_pose)
```

```
# Dance Music Playing Core Logic
def dance_session(self, music_list):
conductor = ConductorAI(tempo_detect_mode="real-time")
for piece in music_list:
# Dynamically generate band score
score = MelodyComposer().arrange(
piece,
orchestra_size=32,
complexity=0.8
)

# Command robot action generation
conductor.load_score(score)
beat_pattern = conductor.analyze_beat()

# Multithreaded execution
processes = [
multiprocessing.Process(target=self.orchestra.play, args=(score,)),
multiprocessing.Process(target=self.dance_robots.perform,
args=(beat_pattern, "waltz"))
```